Antte Vilkko

Kotiautomaatio Raspberry Pi:llä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotekniikka

Insinöörityö

19.10.2015

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Antte Vilkko Kodinohjaus Raspberry Pi:llä 32 sivua 19.10.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Lehtori Ilpo Kuivanen Lehtori Tapio Wikström

Insinöörityössä tutkittiin nykyaikaisia kotiautomaatioteknologioita ja selvitettiin Raspberry Pi -minitietokoneen soveltuvuutta kotiautomaatioon. Tavoitteena oli rakentaa edullinen ja skaalautuva järjestelmä, jota olisi helppo jatkokehittää.

Insinöörityössä rakennettiin modulaarinen laitteistokokonaisuus, jonka keskusyksikkönä oli Pi. Pi:lle toteutettiin palvelinsovellus käyttäen moderneja web-teknologioita, kuten MEAN-pinoa. Kommunikointirajapinta Pi:n ja ulkoisten laitteiden välillä toteutettiin käyttäen infrapuna- ja radioaaltolähettimiä.

Pi osoittautui tehokkaaksi ja monipuoliseksi alustaksi kotiautomaatiotarpeisiin ja se mahdollisti edullisen kotiautomaatiojärjestelmän valmistamisen. Tuotetta on helppo jatkokehittää Pi:n avoimien rajapintojen ja sille julkaistun avoimen lähdekoodin ansiosta.

Avainsanat	Raspberry pi, kotiautomaatio, GCM

Author(s) Title Number of Pages Date	Antte Vilkko HomeAutomation with Raspberry Pi 32 pages 19 October 2015
Degree	Bachelor of Engineering
Degree Program	Information technology
Specialisation option	Software engineering
Instructors	Ilpo Kuivanen, Senior Lecturer Tapio Wikström, Senior Lecturer

The purpose of this thesis is to study modern home automation technologies and to study whether a Raspberry Pi computer can be used in home automation. The aim was to build an inexpensive and scalable system that would be easy to develop further.

Pi was used as the mainframe for a modularized hardware solution. Pi was used for hosting a modern web application built on using modern web technologies, such as MEAN stack. The communication interface between Pi and external devices was carried out using infrared and radio transmitters and receivers.

Pi proved to be an efficient and versatile platform for home automation needs, and it allows manufacturing a low-cost home automation system. The product can be easily further developed, thanks to the open interfaces and the open source code.

Keywords	

Sisällys

Lyhenteet

API	Application Programming Interface. Ohjelmointirajapinnan kautta ohjelmat pystyvät vaihtamaan tietoa keskenään.
DOM	Dokumenttioliomalli, tapa kuvata rakenteisen dokumentin, kuten HTML:n rakennepuuna.
GCM	Google Cloud Messaging, Googlen tarjoama ilmainen palvelu, jolla laitteet ja palvelimet voivat lähettää viestejä Googlen palvelimien välityksellä.
GPIO	General purpose input output, yleiskäyttöinen portti mikro-ohjaimissa, joka voidaan ohjelmoida joko signaalin vastaanottajaksi tai lähettäjäksi.
HTTP	Hypertext Transfer Protocol. Protokolla, jota selaimet ja palvelimet käyttävät tiedonsiirtoon.
I	Ampeeri. SI-järjestelmän mukainen yksikkö sähkövirralle.
JSON	JavaScript Object Notation. Yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen.
MEAN	Yhdistelmä MongoDB-, Express-, AngularJS- ja Node.js -sovelluskehyksistä.
SPA	Single-Page-Application. Sovellus, jonka toiminta on toteutettu yhdellä sivulla.
URL	Uniform Resource Locator. HTTP-protokollan kanssa käytettynä merkeistä muodostuva osoite, josta tietoa voidaan hakea.
V	Voltti. SI-järjestelmän mukainen yksikkö jännitteelle.

1 Johdanto

Insinöörityössä suunniteltiin ja toteutettiin kodinohjausjärjestelmä, jolla käyttäjä pystyy ohjaamaan äänentoistolaitteistoa, valoja, lämmitystä sekä automatisoida palovaroittimen toimintaa.

Työn toteutuksessa käytettiin Raspberry Pi -minitietokonetta kodinohjausjärjestelmän keskusyksikkönä ja sille toteutettiin sovellus, jonka avulla ulkoisia laitteita voi hallita. Työssä selvitetään, kuinka rakennetaan piirilevyjä ja liitetään niitä Pi:n ohjelmoitavaan GPIO:hon. Lisäksi työssä paneudutaan kodinhallintaohjelmiston kehittämiseen käyttäen moderneja web-teknoloioita, kuten Node.js:ää ja AngularJs:ää.

Työssä totetutettiin kolme erinäistä osaa: Pi:n konfigurointi ja piirilevyjen toteutus, verkkosovellus, jolla hallitaan Pi:tä, sekä Android-sovellus, jolla voi vastaanottaa viestejä Pi:ltä ja käyttää verkkosovelluksen käyttöliittymää..

Lopputuloksena syntyi toimiva kodinohjauslaite, jota voi hallita verkkosovellukselta ja Android-sovellukselta. Verkkosovelluksen skaalautuvuuden ansiosta sitä on helppo jatkokehittää ja lisätä ominaisuuksia.

2 Kotiautomaatio

Kotiautomaatio on käsitteenä melko uusi ilmiö, ja siitä puhuttaessa käytetään yleensä termiä *älykoti*. Sen tarkoituksena on helpottaa kodin peruselektroniikan ohjausta, mm. säätää valoja ja lämmitystä. Kotiautomaatio juontaa juurensa rakennusautomatiikasta, jota käytetään lähinnä kerrostaloissa ja isoissa rakennuskomplekseissa, kuten toimistotaloissa tai ostoskeskuksissa. Näissä kohteissa on yleensä tekninen tila, joista on pääsy itse järjestelmään, joka ohjaa tekniikkaa.

Kotiautomaatiossa kodin laitteet liitetään kodin sisäverkkoon, jossa niitä voi ohjata verkkoon kytketyllä laitteella, hallintapaneelilla, tietokoneella tai mobiililaitteella. Myös etäyhteys on mahdollista, riippuen itse kodinohjausjärjestelmästä.

Teknologian kehittyessä ja laitteiden hintojen halpenemisen ansiosta kotiautomaatiosovellukset ovat yleistymässä tavallisten kuluttajien keskuudessa. Nykyään markkinoilla on

useita eri valmistajia, jotka tarjoavat täyden palvelun tuotteita, joilla voi hallita kodin elektroniikkaa ja valvoa kotia kameroin sekä ohjata kodin ovien ja ikkunoiden lukitusta. Nämä tuotteet ovat yleensä suljettuja järjestelmiä ja yhteensopivuus muiden valmistajien laitteisiin on melko olematon. On myös laitteita, jotka tarjoavat vain rajapinnan muiden valmistajien valmistamiin tuotteisiin. Nämä niin sanotut kotiautomaatio-ohjaimet, tarjoavat vain langattomat yhteydet, joilla voi hallita niitä tukevia laitteita.

3 Laitteen toteutus

Työssä oli tarkoitus rakentaa itsenäinen internetin yli toimiva kodinohjausyksikkö, jolla voi hallita ulkoisia laitteita käyttäen infrapunalähettimiä, radiolähettimiä ja sähkövirtaa. Laitteistoalustaksi valittiin Raspberry Pi, sen edullisen hinnan, muunneltavuuden ja ohjelmoitavan GPIOn takia.

Pi:llä on myös laaja yhteisö, joka ylläpitää ja julkaisee uusia sovelluksia sille. Työn toteutuksessa käytettiin yhteisön materiaaleja ja ohjelmia hyödyksi.

3.1 Raspberry Pi

Raspberry Pi on Raspberry Pi Foundationin kehittämä pieni yhden piirilevyn tietokone, joka suunniteltiin olemaan mahdollisimman edullinen, jotta mahdollisimman monella olisi varaa siihen ja näin edistää ihmisten tietoteknistä osaamista. Tietokoneeseen voidaan liittää näyttö, näppäimistö ja hiiri, mutta sitä voidaan käyttää myös ssh-yhteyden yli, sen ollessa liitettynä verkkoon.

Pi:stä on 5 eri versiota tähän mennessä. Kolme aikaisinta versiota käyttää Broadcomin BCM2835 -piirisarjaa, joka sisältää 700 Mhz 32-bittisen ARM prosessorin, Videocore 4 -grafiikkaohjaimen ja 256 tai 512 Mb SDRAM -muistin. Uusin versio sisältää Broadcomin BCM2836 -piirisarjan, joka taas käyttää 900 Mhz neliytimistä ARM Cortex-a7 -prosessoria ja sisältää 1 Gt suuruisen SDRAM-muistin. [1, s. 11-12] Pi toimii kaikilla ARM-pohjaisilla käyttöjärjestelmillä, mutta sitä suositellaan käytettävän Raspbian käyttöjärjestelmän kanssa, joka on optimoitu Pi:tä varten. Se tukee kaikkia ohjelmointikieliä, jotka kääntyvät ARM v6:sella. [2.]

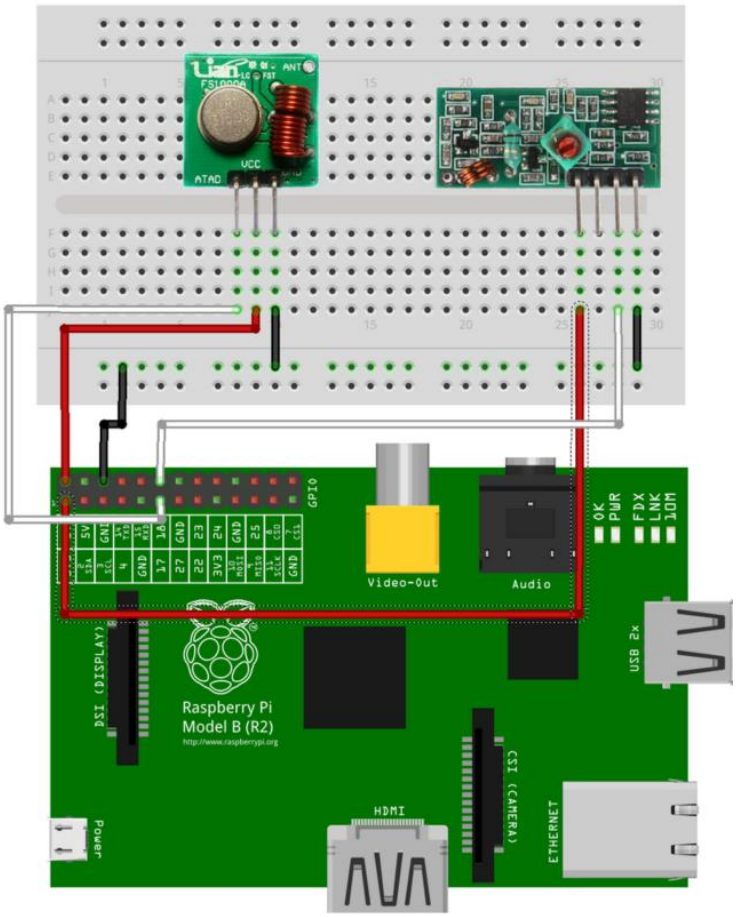
Pi:ssä on myös ohjelmoitava GPIO eli yleiskäyttöinen sisään/ulostulo(General purpouse input/output). Tämä mahdollistaa ulkoisten laitteiden liittämisen Pi:hin ja niiden hallitsemisen Pi:n tarjoaman rajapinnan kautta. GPIO -pinnejä on yhteensä 26 ja näistä 8 ohjelmoitavissa joko sisään- tai ulostuloksi.

3.2 Tarvittavat komponentit

3.2.1 Radiolähetin

Työssä käytettiin Pi:hin kytkettyjä, 433 Mhz:n taajuudella toimivia radiovastaanotin ja -lähetin moduuleita tallentamaan ja toistamaan kaukosäätimen lähettämää käskyä. Lähetintä käytettiin ohjaamaan etäohjattavia pistorasioita, jotka toimivat myös 433 Mhz:n taajuudella. Jotta radiolähettimellä pystyttiin lähettämään signaali pistorasialle, pistorasian oman kaukosäätimen lähettämä signaali täytyi ensin tallentaa radiovastaanottimella.

Lähetin ja vastaanotin toimivat oletuksena 5 V:n jännitteellä. Lähetintä ei kuitenkaan voida kytkeä Pi:n 5 V:n jännitelähteeseen, koska se ohjaa jännitteen Pi:n datasisääntuloon, jonka maksimijännitteenkesto on 3,3 V. Tämän takia lähetin kytkettiin Pi:n 3,3 V:n jännitelähteeseen. Työssä valittiin käytettäviksi dataportti 16 signaalin lähettämiseksi ja vastaanottamiseksi 11. Moduuleissa oli selvät merkinnät virran ulos- ja sisäänmenoille, joten ne kytkettiin Pi:n virransyöttöön ja maahan.



Kuva 1. Radiomoduuleiden liittäminen Rasbperry Pi:hin.

Raspberry Pi:lle löytyi valmis ohjelma rcswitch-pi, joka pystyy lukemaan analogisesti kaappaamansa signaalin ja lähettämään sen. Rcswitch-pi käyttää WiringPi-nimistä kirjastoa toimiakseen, joten sekin täytyy asentaa. Ohjelman asennus oli suoraviivaista:

rcswitch-pi ohjelman asennus:

```
git clone git://github.com/ninjablocks/433Utils.git
cd 433Utils/RPi_utils/
```

wiringPI ohjeman asennus ja koonti

```
git clone git://git.drogon.net/wiringPi
cd wiringPi
./build
```

Signaalin lukemiseksi ohjelma täytyi käynnistää:

```
sudo ./RFSniffer
```

Nyt kaukosäätimen painiketta painaessa konsolille tulostui

Received 1135923

Tämä on analoginen arvo kaukosäätimen lähettämälle signaalille. Kun koodi on tiedossa, sen voi lähettää ohjelmalla.

./codesend 1135923

Pistorasia ei kuitenkaan reagoinut signaaliin ja tutkimisen jälkeen kävi ilmi, että signaali on pidempi, kuin mitä rcswitch-pi oletuksena toistaa. Tämän takia täytyi tallentaa myös signaalin pituus ja tehdä muutoksia rcswitch-pi-sovellukseen, jotta signaali toistetaan oikean pituisena. Muutosten jälkeen pistorasia reagoi signaaliin.

3.2.2 Infrapunalähetin

Projektissa käytettiin infrapunalähetintä ja vastaanotinta tallentamaan ja toistamaan kaukosäätimen signaalia. Tarkoituksena oli tallentaa kaukosäätimien eri painikkeiden signaalit ja lähettää näitä signaaleja raspilla. Signaalin lukemiseen ja lähettämiseen on valmis kirjasto LIRC.

Infrapunalähetin toimii 100 mA:n virralla ja 1.6 V:n kynnysjännitteellä, ja infrapunapulssin vaatima virran suuruus on 1A [3.]. Pi:stä saatava kantajännite on vain 16 mA, joten väliin tarvittiin transistori toimimaan kytkimenä, jotta saadaan tarvittava virta. Transistoria varten tarvittiin myös etuvastus laskemaan sille saapuvaa jännitettä.

Jotta virtapiiri toimisi, tarvittiin transistori, joka kestää vähintään 100 mA:n virran ja 1,6 V:n jännitteen. Tähän valittiin 550BC NPN-transistori, joka täyttää kriteerit. Lähettimen etuvastuksen laskemiseksi käytettiin 100 mA:n virtaa ja vastuksen arvo saadaan seuraavasta kaavasta:

$$\frac{U}{I} = R \qquad (1)$$

U on *GPIO*:n 3,3 V – lähettimen kynnysjännite 1,6 V
I on GPIO:n *jännite*

$$\frac{3{,}3\ V - 1{,}6\ V}{0{,}1\ I} = 17\ \Omega \qquad (2)$$

Transistorin kantavastuksen laskemiseksi täytyi selvittää transistorin minimivahvistus, joka löytyi transistorin datasivulta (datasheet) [4, s. 2]. Koska haluttu ohjausvirta oli tiedossa (100 mA), niin kantavastuksen laskeminen kävi seuraavasti. Haluttu ohjausvirta / minimi vahvistus = minimi kantavirta

$$\mathrm{Ic(max)} \,/\, \mathrm{hFE(min)} \;=\; \mathrm{IB(min)} \qquad (3)$$

Ic(max) on haluttu ohjausvirta
hFE(min) on minimivahvistus
IB(min) on minimi kantavirta

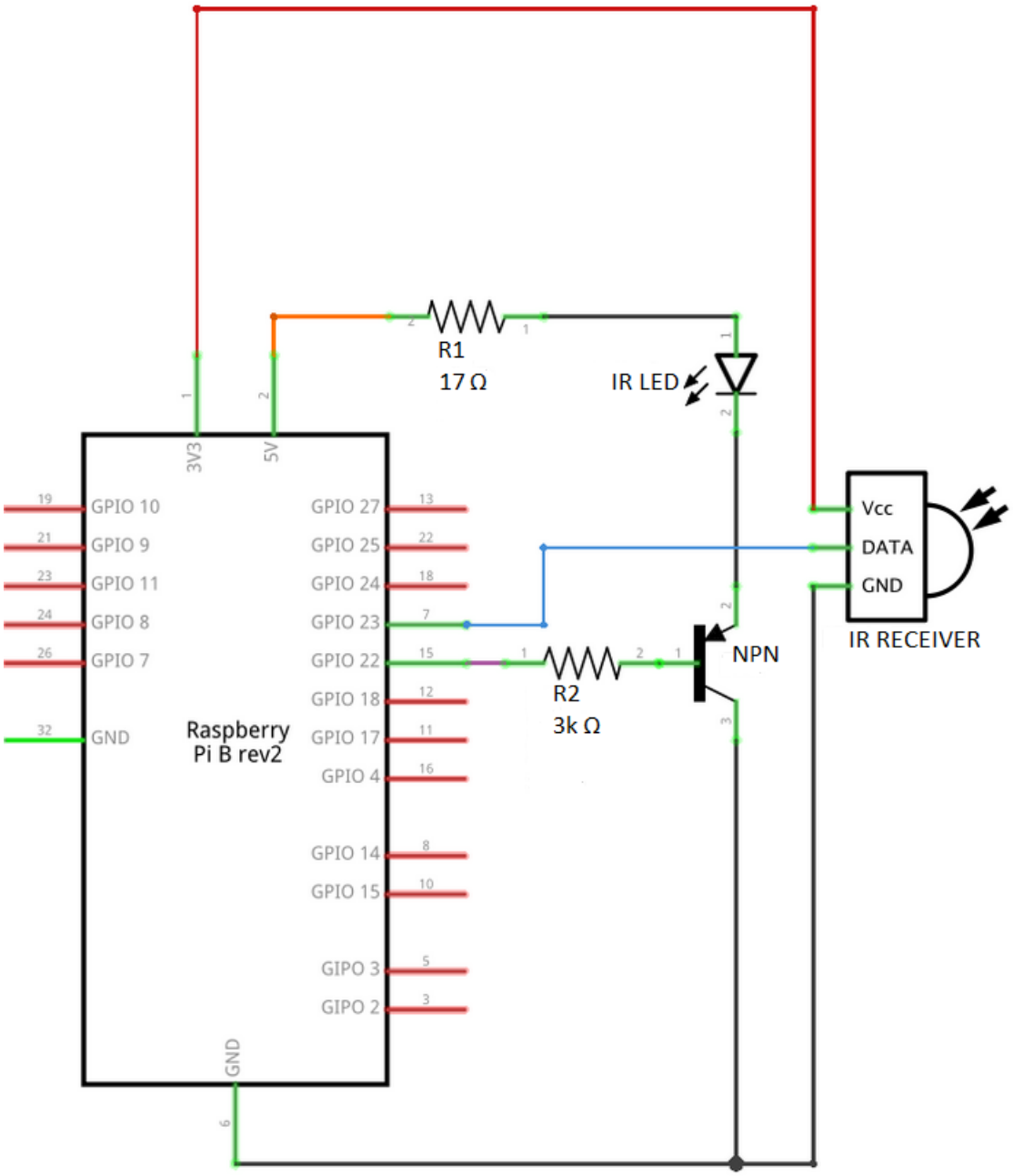
joten 100 mA / 110 mA = 0,9 mA

Kun minimikantavirta oli selvillä, pystyttiin laskemaan transistorin vaatiman etuvastuksen suuruus. GPIO-portin ohjausjännite on 3,3 V ja transistorin kynnysjännite on 0,6 V, joten vastuksen suuruus on:

$$\frac{3.3V-0.6V}{0.9mA} = 3000\ \Omega \qquad (4)$$

Etuvastuksena täytyy siis käyttää 3000 ohmin suuruista vastusta.

Vastaanottimen dataportti liitettiin GPIO:n datasisääntuloporttiin 23 ja lähettimen dataportti liitettiin GPIO:n dataulostuloporttiin 22. Tämän jälkeen virtapiiri oli valmis käytettäväksi ja infrapunakomponentteja voi hallita LIRC-ohjelmalla.



Kuva 2. Infrapuna-antureiden kytkentäkaavio.

LIRC:n asentaminen käy seuraavasti:

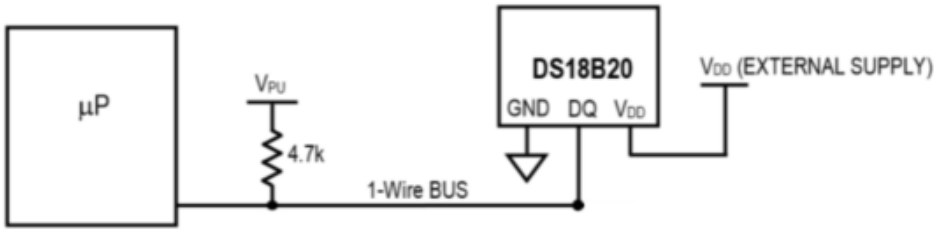
```
sudo apt-get install lirc
```

Tämän jälkeen ohjelmalla voi alkaa tallentamaan kaukosäätimen signaaleja. Ohjelman tallentaessa painalluksia se pyytää jokaisen painalluksen jälkeen näppäimen nimen, joka

tallennetaan konfiguraatiotiedostoon yhdessä signaalin kanssa. Kun signaalit on tallennettu, niitä voi toistaa komennolla irsend SEND_ONCE ”konfiguraatiotiedoston nimi” ”painikkeen nimi”.

3.2.3 Lämpötila-anturi

Lämpötila-anturina käytettiin ohjelmoitavaa DS18B20 digitaalista lämpötilamittaria. Anturi käyttää yksijohtoista rajapintaa kommunikointiin, 3,3 V:n jännitteellä. Anturin datalehdellä [5, s. 6] oli komponentin kytkentäkaavio, jota käytettiin apuna komponenttia raspiin kytkettäessä. Datalehden kytkentäkaavion mukaisesti anturi kytkettiin Pin:n 3,3 V:n ulostuloon ja sisääntuloporttiin 7.



Kuva 3. Esimerkkikytkentäkaavio.

Kun komponentti oli kytketty, piti ladata kaksi Linuxin kernel -moduulia, jotka käynnistivät anturin rajapinnan.

sudo modprobe w1-gpio && sudo modprobe w1-therm

Tämän jälkeen Pi tunnisti anturin ja sen lämpötilan pystyi lukemaan sys/bus/w1/devices/ -hakemistosta. Hakemistoon generoitui kansio, jonka nimi vastasi lämpötila-anturin yksilöivää nimeä. Kansion sisällä oli w1_slave niminen -tiedosto, johon lämpötila kirjoitettiin. [6, s. 40-51.]

3.2.4 Palovaroitin

Palovaroitin oli normaaleilla pattereilla toimiva malli, johon liitettiin ohjausvirtajohto. Ohjausvirtajohto kytkettiin samaan porttiin palovaroittimen äänilähteen kanssa, ja toinen

pää kytkettiin Pi:n sisääntuloporttiin 25. Kun palovaroitin havaitsee savua, se laukaisee hälytyksen, jolloin virta kulkee äänilähteelle ja Pi:lle.

Ohjausvirtajohtoon täytyi liittää vastus, koska Pi ei kestä patterin 9 V:n jännitettä. Vastus laskettiin käyttämällä GPIO:n maksimimaalista virrankestoa 0,5 mA.

$$\frac{9\ V-3{,}3\ V}{0{,}5\ mA} = 11400\ \Omega \quad (5)$$

Raspberry Pi:lle toteutettiin kuuntelija, joka kuuntelee sisääntuloportin tilaa. Jos portin tila muuttuu, eli sille alkaa virrata virtaa niin kuuntelija laukaisee prosessin, joka kutsuu Google Cloud messaging -palvelua. Gcm:stä ja sen toteutuksesta lisää luvussa 4.1.1 ja 4.2.2.

4 Sovellus

4.1 Toteutettujen sovelluksien yleiskuvaus

Sovellus toteutettiin käyttäen MEAN-pinototeutusta, joka tulee sanoista MongoDB, Express.js, Angular.js ja Node.js. MEAN-pinon sovellukset toimivat tunnetusti hyvin yhdessä, josta vakiintunut MEAN-lyhennelmäkin on syntynyt. [7]

AngularJS:llä toteutettiin responsiivinen yksisivuinen (single-page app) asiakassovellus. Asiakassovelluksen ja Node.js-palvelinsovelluksen kommunikointirajapinta toteutettiin REST-arkkitehtuurimallin mukaisesti.

Tavoitteena oli myös luoda helppokäyttöinen Android-sovellus, jolla pystyy käyttämään asiakassovelluksen käyttöliittymää. Tiedonsiirron helpottamiseksi työssä käytettiin Google Cloud Messaging -palvelua, joka tarjoaa rajapinnan sovelluspalvelimen ja Android-sovelluksen väliseen kommunikointiin. Tämän ansiosta viestit, jotka sovelluspalvelin lähettää Android-sovellukselle, menevät perille, vaikka Android-laite olisi suljettu.

4.1.1 Google Cloud Messaging

Google Cloud Messaging on Googlen tarjoama palvelu, jonka avulla omalta palvelimelta voi lähettää viestejä Android-sovellukseen Googlen pilvipalvelun kautta. Palvelun etuna on sen yksinkertaisuus ja keveys. GCM huolehtii automaatiisesti kaikesta viestin lähettämiseen liittyvistä asioista, kuten viestien jonotuksesta ja niiden toimittamisesta päätelaitteelle.

GCM mahdollistaa myös viestien lähettämisen Android-laitteelta omalle palvelimelle Googlen pilvipalvelun kautta, mutta tässä työssä ei käytetty sitä, joten siihen aiheeseen ei kestitytetä.

GCM:n toteuttamiseksi vaaditaan vähintään:

- Android-sovellus
    - Vähintään Android versio 2.2, Google Play Store asennettuna ja Google tili, tai Android versio 4.0
- Googlen GCM-palvelin
    - Palvelin ottaa vastaan viestejä omalta palvelimelta. Tätä varten täytyy rekisteröityä google kehittäjäksi ja maksaa 25$ kehittäjämaksu.
- Oma palvelin
    - Kommunikoi GCM-palvelimen ja Android-sovelluksen kanssa.

GCM-palvelu tarjoaa kaksi eri yhteysprotokollaa, jolla palvelua voi käyttää: HTTP ja XMPP, niitä voi käyttää joko rinnakkain tai erikseen. HTTP-protokollaa käytettäessä palvelu mahdollistaa viestien lähettämisen vain sovelluspalvelimelta Android-sovellukselle, kun taas XMPP tarjoaa tuen myös viestien lähettämiseen Androidilta sovelluspalvelimelle. Koska tässä työssä ei nähty olennaiseksi lähettää viestejä Android-sovellukselta sovelluspalvelimelle käyttäen GCM:ää, päätettiin käyttää HTTP -protokollaa. HTTP-protokollaa käytettäessä viestit lähetetään synkronisesti, jolloin sovelluspalvelin lähettää viestin http POST-pyynnön ja jää odottamaan vastausta. Tämän takia sovelluspalvelin ei voi lähettää uutta viestiä ennen kuin edelliseen POST-pyyntöön on vastattu. [8]

HTTP-protokolla lähettää viestit joko normaalina tekstinä, tai JSON-tyyppisinä objekteina, jossa data on sisällytetty viestiin avain/arvo parina. Esimerkkinä JSON-objekti jossa avaimena nimi-kenttä ja arvona nimi:

```
{  "nimi": "Antte"  }
```

GCM-palvelu käyttää seuraavia yksilöllisiä tunnistetietoja päätelaitteen ja palvelimen tunnistamiseen.

- Sender ID
    - Uniikki id, joka luodaan Google Developer Consolessa. Tätä käytetään kirjautumisprosessissa, tunnistamaan palvelin, jolla on oikeus lähettää viestejä Android-sovellukseen.
- API Key
    - Tallennetaan palvelimelle, jolle annetaan oikeus käyttää Googlen palveluja. HTTP-protokollaa käytettäessä API Key sisällytetään http POST-pyynnön otsakkeisiin.
- Application ID
    - Android-sovelluksen nimi. Tämän avulla sovellus rekisteröidään, jotta se voi vastaanottaa viestejä.
- Registration Token
    - GCM-palvelimen lähettämä id, jonka ansiosta Android-sovellus voi vastaanottaa viestejä.

Android-sovelluksessa täytyy tehdä seuraavat konfiguraatiomuutokset, jotta GCM toimisi. Android-projektiin täytyy lisätä projektikohtainen asetustiedosto, jonka voi ladata Google Developers Consolesta. Lisäksi projekti pitää asettaa käyttämään Google Play -palvelua ja sovelluksen manifest-tiedostoon täytyy lisätä seuraavat oikeudet. [9]

- Android.permission.Internet
    - Tämä antaa sovellukselle oikeuden käyttää internetiä.
- Android.permission.WAKE_LOCK

    - Tämän ansiosta sovellus pystyy pitämään prosessorin valvetilassa, kun viesti saapuu.

- com.google.anroid.c2dm.permission.RECEIVE

    - Antaa sovellukselle oikeuden rekisteröityä ja vastaanottaa viestejä.

- com.google.android.c2dm.permission.SEND

    - Antaa sovellukselle oikeuden lähettää viestejä.

- ”Sovelluspaketin nimi” + .permission.C2D_MESSAGE

    - Estää muita Android sovelluksia rekisteröitymästä ja vastaanottamasta viestejä.

- GcmReceiver-luokan esittely.

    - GcmReceiver huolehtii GCM-palvelimelta saapuvien viestien vastaanottamisesta

- GcmListenerService luokan esittely

    - Huolehtii viestien käsittelystä.

- InstanceIDListenerService palvelun esittely.

    - Huolehtii Registration ID:n luomisesta ja päivittämisestä.

GCM:än implementointi omalle sovelluspalvelimelle.

Omalla sovelluspalvelimella pitää olla tiedossa Registration token ja Sender ID, jotta se pystyy lähettämään viestejä GCM-palvelimelle. Tätä varten sen täytyy pystyä vastaanottamaan http POST -pyyntö Android-sovellukselta, jossa nämä tiedot välitetään.

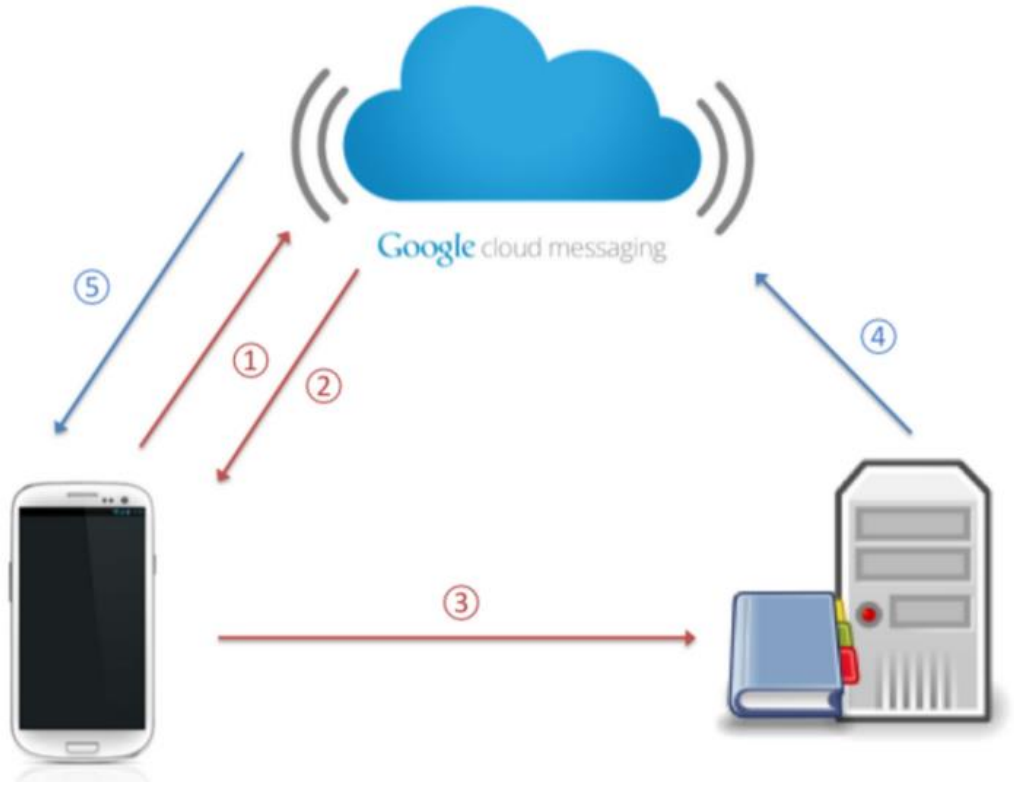
Kun Registration ID ja Sender ID ovat tiedossa, sovelluspalvelin voi lähettää http POST-pyynnön GCM-palvelimelle. Jotta GCM-palvelin osaa käsitellä viestin, se täytyy rakentaa API:n mukaisesti. Http-otsakkeen täytyy sisältää kaksi eri osaa. Authorization-osa, joka välittää Sender ID:n ja Content-Typen, joka kertoo viestin sisällön käyttämän merkintätyypin. Merkintätyyppi voi olla joko JSON-tyyppinen tai selkokielinen (plain-text). HTTP-body sisältää viestin sisällön ja Registration IDn.

```
Content-Type:application/json
Authorization:key=AIzaSyZ-1u...0GBYzPu7Udno5aA

{
  "to" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
  "data" : {
    ...
  },
}
```

Kuva 4. Sovelluspalvelimen lähettämän http POST -pyynnön rakenne.

GCM:n elinkaari



Kuva 5. GCM-toteutuksen toimintaperiaate.

1. Android-sovellus lähettää sender id:n GCM-palvelimelle ja rekisteröityy vastaanottamaan viestejä.

2. GCM-palvelin lähettää Registration Tokenin Android-sovellukselle.

3. Android-sovellus lähettää vastaanottamansa Registration Tokenin sovelluspalvelimelle. Sovelluspalvelin tallentaa Registration Tokenin, jonka jälkeen se on valmis lähettämään viestejä GCM-palvelimelle.

4. Sovelluspalvelin lähettää viestin GCM-palvelimelle.

5. GCM-palvelin lähettää viestin Android-sovellukselle.

4.1.2 Node.js

Node.js on alusta palvelinovelluksille, joka on rakennettu Google V8 JavaScript -moottorin päälle. Node.js-sovellukset on kirjoitettu JavaScriptillä ja niitä voidaan ajaa Node.js:llä useilla eri käyttöjärjestelmillä.

Node.js käyttää tapahtumapohjaista arkkitehtuuria ja ei-estävää I/O APIa, joka on suunnisteltu optimoimaan sovelluksen datan käsittely reaaliaikaisille sovelluksille. Tapahtumapohjaisuus mahdollistaa skaalautuvien web-palvelimien kehittämisen, käyttämällä takaisinkutsua (callback) säikeiden sijaan. Node.js sisältää kirjaston, joka mahdollistaa sen toimivan web-palvelimena ilman ylimääräistä ohjelmistoa, kuten Apachen HTTP-palvelinta.

Npm on Node.js:n käyttämä paketinhallintajärjestelmä, joka asennetaan automaattisesti Node.js:n asennuksen yhteydessä. Se on komentorivisovellus, jolla hallitaan sovelluksen moduulien riippuvuuksia. Sillä voi myös asentaa Node.js -sovelluksia, jotka ovat saatavilla npm:n julkisessa kokoelmassa.

Moduulin asentamisessa käytetään npm install ”paketin nimi” -komentoa. Asennuksen yhteydessä voi antaa g -parametrin, jolloin se asennetaan globaalisti, eli sitä voidaan käyttää esimerkiksi komentoriviltä. Mikäli asennettavaa sovellusta halutaan käyttää omassa Node.js-sovelluksessa käyttäen ’require (’paketin nimi’) tyyliä, se asennetaan lokaalisti ilman -g parametria.

```
npm install forever -g
```

Kuva 6. Forever-moduulin asentaminen globaalisti.

4.1.3 ExpressJS

ExpressJS on kevyt web-ohjelmistokehys, joka helpottaa sovelluksen organisoimista mvc-suunnittelumallin mukaisesti. ExpressJS pohjautuu Node.js:n http-moduuliin ja Connect-komponenttiin. Se sisältää ominaisuuksia, joiden ansiosta kaikkea toiminnollisuutta ei tarvitse koodata. ja se tarjoaakin komponentteja, joilla voi helposti implementoida esim. sovelluksen reitityksen ja sessionhallinnan. [11, s. 3-4.]

4.1.4 MongoDB

MongoDB on kirjoitushetkellä suosituin dokumenttipohjainen tietokantajärjestelmä, joka pohjautuu NoSQL:ään. NoSQL (non SQL) on skaalautuva tietokanta, joka ei perustu perinteiseen relaatiomalliin. Relaatiomallittomuus tekee NoSQL:n skeemasta dynaamisen, joten skeemaa ei tarvitse määrittää ennen datan syöttämistä. Dynaamisen skeeman ansiosta tietomalliin ei tarvitse tehdä muutoksia, vaikka syötetyn datan malli muuttuisi. Esimerkiksi verkkokaupassa kaikilla myytävillä tavaroilla on hinta, mutta niiden ominaisuudet eroavat toisistaan.

```
{
  id: "00e8da9b",
  type: "Audio Album",

  pricing: {
    list: 1200,
    retail: 1100
  },

  details: {
    title: "The Album",
    artist: "Haddaway",
    genre: [ "Dance", "Eurodance" ],
        ...
    tracks: [
      ""What is love",
      ""Shout" ",
      ...
    ],
  },
}
```

```
{
  id: "00e8da9d01",
  type: "Film",

  pricing: {
    list: 1500,
    retail: 1000
  },

  details: {
    title: "The Matrix",
    director: [ "Andy Wachowski", "Larry Wachowski" ],
    writer: [ "Andy Wachowski", "Larry Wachowski" ],
    ...,
    aspect_ratio: "1.66:1"
  },
}
```

Kuva 7. Havainnekuva kahdesta tavaratyyppisestä dokumentista.

MongoDB tallentaa tiedon dokumenttityyppisesti, jossa yksittäinen dokumentti sisältää avain/arvopareja. Dokumenttia voidaan verrata relaatiomallin taulun riviin, jossa avain/arvopari vastaa rivin sarakenimeä ja sen arvoa. Kuvassa 7 details -avain pitää sisällään siihen liittyvät arvot. [12, s. 4-6.]

4.1.5 AngularJS

AngularJS on Google ylläpitämä avoimeen lähdekoodin perustuva JavaScript -ohjelmistokehys dynaamisten web -sovellusten kehittämiseen. Se on kasvattanut suosiotaan viimeisten vuosien aikana ja sille on kehittynyt laaja kehittäjien verkosto. AngularJS:n uusin version on 1.4 ja versio 2.0 julkaistaan todennäköisesti kesällä 2016.

AngularJS perustui alun perin MVC-arkkitehtuuriin(Model-View-Controller, Malli-Näkymä-Kontolleri)). Näkymää edustaa DOM, kontrollerit ovat JavaScript-funktioita, ja malli on data. Vuosien saatossa siitä on kehittynyt kuitenkin enemmän MVW-arkkitehturiaalinen ohjelmistokehys. Model-View-Whatever -arkkitehtuuri ei lokeroidu mihinkään tiettyyn arkkitehtuurimalliin, vaan sen on ajateltu toimivan ”What ever works for you” -mallisesti.

Perinteisissä web-sovelluksissa sovelluksen näkymä luodaan palvelimella, jossa haluttu data sisällytetään HTML-sivuun. Aina kun data muuttuu, tai sitä halutaan päivittää, koko sivu täytyy ladata uudestaan. Tämä luo runsaasti turhaa liikennettä ja syö näin resursseja. AngularJS on ratkaisut tämän ongelman kaksisuuntaisella datakytkennällä, jossa käyttöliittymälle lähetetään vain data, joka on muuttunut.

Perinteinen Hello Wold -sovellus AngularJS:llä näyttää seuraavalta.

```
<!DOCTYPE html>
<html ng-app>
<head>
Kirjoita nimesi:
    <input type="text" ng-model="nimi" />

    <h1>Hello {{ nimi }}</h1>
</body>
</html>
```

Kuva 8. AngularJS Hello World -sovellus.

Sovelluksessa määritellään attribuutti ng-model, joka sitoo tekstilaatikon tilan ng-modelin arvoon eli muuttujaan ”nimi”. Kun tekstilaatikon teksti muuttuu, Angular muuttaa nimi muuttujan arvoa, ja jos arvo muuttuu, niin tekstilaatikon teksti muuttuu. Tätä kutsutaan kaksisuuntaiseksi datakytkennäksi.

Direktiivit ovat tunnisteita DOM-elementissä, jotka kertovat AngularJS:n HTML-kääntäjälle (compiler), että kyseiseen DOM-elementtiin halutaan kytkeä tietty toiminnollisuus.

AngularJS:ssa on useita valmiita direktiivejä, kuten ngBind, ngApp ja ngModule. Niitä voi myös luoda ja käyttää HTML-elementtien tapaan. Esimerkkinä Hello World -direktiivi.

```
var app = angular.module('esimerkki', []);

app.directive('helloWorld', function() {
  return {
      restrict: 'AE',
      template: '<h3>Hello World!!</h3>'
  };
});
```

Kuva 9. Hello World -direktiivi.

Kuvassa 9. app.directive() rekisteröi uuden direktiivin moduuliin. Ensimmäinen argumentti on uuden direktiivin nimi ja toinen argumentti on funktio, joka palauttaa Hello World HTML-elementin. Restrict arvo määrittää, kuinka direktiiviä käytetään HTML:ssä. Arvo ”AE” kertoo, että sitä voidaan käyttää attribuuttina tai elementtinä. Template arvo määrittää HTML:n, joka liitetään näkymään, kun AngularJS on kääntänyt direktiivin. Tämän jälkeen direktiiviä voi käyttää HTML:ssä normaalin HTML:n tapaan.

```
<hello-world/>
```

Kuva 10. helloWorld direktiivin käyttö HTML:ssä.

Riippuvuusinjektio on suunnittelumalli, jossa komponentille annetaan sen tarvitsemat riippuvuudet ilman niiden kovakoodausta. Tämän ansiosta riippuvuudet ovat konfiguroitavissa, eikä komponentin tarvitse tietää, mistä riippuvuudet tulevat.

AngularJS sisältää valmiin riippuvuusinjektio mekanismin, jonka ansiosta AngularJS sovellus on helppo jakaa komponentteihin, joita voi injektoida toisiinsa. Sovelluksen modularisoinnin ansiosta komponentteja on helpompi käyttää uudelleen, konfiguroida ja testata.

```
var app = angular.module("DemoApp", []);
app.controller("DemoController", function ($scope, $http) {
    $http.get('https://api.github.com/users/angular/repos')
        .success(function (repos) {
        $scope.repos = repos
    });
});
```

Kuva 11. Riippuvuusinjektio AngularJS:ssä

Kuvassa 11 kontrolleriin DemoController injektoidaan $scope- ja $http-objektit. Kontrolleri ei tiedä, kuinka nämä objektit luodaan. Se vain tietää, kuinka käyttää niitä.

4.2 Kodinohjaussovelluksen toteutus

MEAN-ohjelmistopinon kokoaminen aloitettiin sovelluksen rautalankamallin pystyttämisellä. Tähän oli saatavilla npm-sovellus, express-generator, joka luo sovellukselle kansiorakenteen ja HTTP-palvelimen.

Angularin toiminnollisuuden sitominen (bootstrap) näkymään tehdään ngApp-direktiivillä. NgApp määrittää Angular-sovelluksen juurielementin, jotta Angularin koostaja (compiler) osaa koostaa sen sisällä olevat Angular-elementit. Käyttöliittymän tilan hallintaan käytettiin ui-router moduulia. Ui-routerilla on mahdollista luoda näkymiä käyttöliittymään, jotka injektoidaan siihen ajonaikana. Tätä lähestymistapaa käytettiin siksi, että sovellusta olisi helppo ylläpitää ja siihen olisi helppo lisätä uusia näkymiä tulevaisuudessa. Näkymät määritellään ui-routerin stateProvider metodilla, jossa haluttu näkymä ja vastaava URL-osoite kytketään yhteen.

Koska kodinohjaussovellus on yksisivuinen sovellus, luotiin vain yksi main-näkymä ja kontrolleri. Kontrollerilla ja näkymällä on kummallakin pääsy jaettuun scope-objektiin, joten scope toimii linkkinä niiden välillä.

```
var app = angular.module('myApp', ['ui.router']);

app.config(function ($stateProvider, $urlRouterProvider) {
    //
    // For any unmatched url, redirect to /
    $urlRouterProvider.otherwise("/");

    $stateProvider
        .state('main', {
            url: "/",
            templateUrl: "/partials/main.html",
            controller: function ($scope, $http, $sce) {

                $scope.power = function () {
                    $http({
                        method: 'GET',
                        url: '/tuner/power'
                    }).success(function (response) {});
                }
```

Kuva 12. Ui-routerin määritys ja power-funktion toteutus.

Kuvan 12. power()-metodin sisällä luodaan http GET-pyyntö, osoitteeseen ”/tuner/power/”. Kun http-pyyntö lähetään, niin ExpressJS:n reitittäjä(router) tutkii, onko sille vastaavaa URL:ia. Mikäli vastaava URL löytyy, niin reitittäjä sieppaa URL:in ja suorittaa siihen liitetyn toiminnollisuuden.

```
app.get('/tuner/power', isLoggedIn, function (req, res) {
    exec('irsend SEND_ONCE SONY_RM-AAU014 BTN_POWER', function () {});
});
```

Kuva 13. ExpressJS:n reititys ’/tuner/power/’ URLI:lle

Havainnekuvassa ’/tuner/power/’ kutsuttaessa suoritetaan shell-skripti, joka kutsuu irsend-komentoa. Irsend-komento lähettää aiemmin määritellyn infrapunakomennon laittaakseen kotiteatterijärjestelmän päälle.

Kaikkien painikkeiden toiminta on toteutettu samalla tavalla. Scope-objektissa määritellään painikkeiden toiminnollisuudet, jotka lähettävät palvelimelle http-pyyntöjä, jossa ne käsitellään. Näin toiminnollisuus on jaettu moduuleihin testattavuuden ja ylläpidon helpottamiseksi.

4.2.1 Autentikoituminen

Kuvassa 15 kutsuttiin funktiota isLoggedin, joka tarkastaa, onko käyttäjä autentikoitunut. Autentikointi implemoitiin käyttämällä MongoDb:tä ja passport-moduulia. Passport konfiguroitiin käyttämään lokaalia autentikoitumista, jossa käyttäjätunnukset on tallennettu MongoDB:hen.

Käyttäjätunnusten tallentamista varten luotiin User-skeema, jossa määriteltiin käyttäjälle käyttäjätunnus ja salasana, joka tallennetaan kryptattuna.

```
var mongoose = require('mongoose');
var bcrypt   = require('bcrypt-nodejs');

// define the schema for our user model
var userSchema = mongoose.Schema({

    local            : {
        username     : String,
        password     : String,
    },
});

// methods ======================
// generating a hash
userSchema.methods.generateHash = function(password) {
    return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null);
};

// checking if password is valid
userSchema.methods.validPassword = function(password) {
    return bcrypt.compareSync(password, this.local.password);
};

// create the model for users and expose it to our app
module.exports = mongoose.model('User', userSchema);
```

Kuva 14. User-skeema.

Käyttäjäskeeman luomisen jälkeen luotiin LocalStrategy, joka käyttää käyttäjäskeemaa käyttäjän autentikoimisen validoimiseksi. Kun isLoggedin funktoita kutsutaan, se välittää LocalStrategylle käyttäjätunnukset ja LocalStrategy kutsuu käyttäjäskeeman validPassword-funktiota, joka tarkistaa, löytyykö käyttäjä tietokannasta. Mikäli käyttäjä löytyy, se palautetaan kutsuvalle funktiolle, ja autentikoituminen on valmis.

```
passport.use(new LocalStrategy({
        usernameField : 'username',
        passwordField : 'password',
        passReqToCallback : true
    },
    function(req, username, password, done) {
        if (username)
            username = username.toLowerCase();
        // asynchronous
        process.nextTick(function() {
            User.findOne({ 'local.username' :  username }, function(err, user) {
                // if there are any errors, return the error
                if (err)
                    return done(err);

                // if no user is found, return the message
                if (!user)
                    return done(null, false, req.flash('loginMessage', 'No user found.'));

                if (!user.validPassword(password))
                    return done(null, false, req.flash('loginMessage', 'Oops! Wrong password.'));

                // all is well, return user
                else
                    return done(null, user);
            });
        });
```

Kuva 15. LocalStrategyn toteutus.

IsLoggedIn funktiokutsu täytyy sisällyttää jokaiseen kutsuttavaan reittiin, jotta vain autentikoitunut käyttäjä pystyy käyttämään sovellusta.

4.2.2 Google Cloud Messaging-palvelun implementointi

Sovelluksen käynnistymisen yhteydessä käynnistetään kuuntelija palovaroittimen varoitussignaalille. Kuuntelijan toteutuksessa käytettiin onoff-moduulia, joka tarkkailee sisääntulosignaalin tilan muuttumista. Normaalitilanteessa sisääntulosignaali on 0 V. Tilan muuttuessa kuuntelija kutsuu gcm-moduulia, joka lähettää viestin sovellukselle rekisteröityneelle Android-laitteelle.

```
var gcm = require('node-gcm');
var Token = require('../models/token');

module.exports = function(){
    var message = new gcm.Message();

        message.addData('Fire!', 'raspb');
    var regIds = [];
    Token.findOne({}, {}, {sort:{'created_at':-1}}, function(err, token) {
        if(err)
            return console.log(err);
        if(!token)
            return console.log('token not found');
        regIds.push(token.token);

        // Set up the sender with you API key
        var sender = new gcm.Sender('AIzaSyCLveIqP3Qn15jD6dBaXJW2llzuz-tpcJs');

        // ... or retrying a specific number of times (10)
        sender.send(message, { registrationIds: regIds }, 3, function (err, result) {
          if(err) console.error(err);
          else    console.log(result);
        });

    });
}

```

Kuva 16. Gcm-moduulin toteutus.

Gcm-moduulin toteutuksessa käytettiin node-gcm -pakettia, jolla voidaan lähettää gcm viestejä. Toteutetussa moduulissa tarvittava token haetaan tietokannasta. Tietokannassa voi olla monta tokenia, koska Android-sovellus lähettää automaattisesti uuden tokenin node-palvelimelle, jos token on päivittynyt. Tämän takia tietokannasta haetaan aina uusin token, jota käytetään yhdessä API-keyn kanssa lähetettäessä viestiä.

4.2.3 Herätystoiminnon toteutus

Herätystoiminnossa luodaan ajastettuja cron-toimintoja, jotka suoritetaan käyttäjän määrittelemänä aikana. Käyttäjä voi syöttää ajan käyttöliittymällä klikkaamalla kytkintä (switch) Alarm -osiossa. Tämä avaa ajansyöttö ponnahdusikkunan, johon ajan voi syöttää. Kun käyttäjä on syöttänyt ajan, se välitetään crontab-moduulille, jossa luodaan ajastettu komento. Komento suorittaa skriptin, joka laittaa valot ja musiikin päälle.

Kun Angular luo main-näkymän, se kutsuu cronia, joka tarkastaa, onko käyttäjä luonut herätys-tehtävän. Jos tehtävä löytyy, niin käyttöliittymän Alarm-kytkin laitetaan aktiiviseen tilaan ja ajansyöttötekstikenttään asetetaan tehtävälle määrätty suoritusaika. Käyttäjä voi myös perua herätyksen klikkaamalla kytkintä sen ollessa aktiivisessa tilassa.

Tämä poistaa herätys cron -tehtävän. Seuraavan luvun lämmitystoiminnon käyttöliittymä -näkymä toteutettiin samalla tavoin.

4.2.4 Lämmitystoiminnon toteutus

Lämmistystoiminnon toteutuksessa käytettiin lämpömittaria, lämpöpatteria ja kauko-ohjattavaa pistorasiaa. Kun käyttäjä syöttää halutun lämpötilan käyttöliittymältä, sovellus tallentaa lämpötilan tietokantaan. Tämän jälkeen luodaan cron-tehtävä, joka suoritetaan viiden minuutin välein. Tehtävä tarkistaa viiden minuutin välein lämpömittarin lämpötilan ja vertaa sitä tietokannassa olevaan. Kun haluttu lämpötila on saavutettu, niin lämpöpatteri sammutetaan. Lämpötilan laskiessa alle käyttäjän määrittelemän lämpötilan, lämpöpatteri laitetaan takaisin päälle. Näin pyritään säilyttämään tasainen lämpötila.

```
var sensor = require('ds18x20');
var Temperature = require('../models/temperature');

Temperature.findOne({}, function (err, desiredTemp) {
    if (err)
        return console.log(err);
    sensor.get(function (err, temp) {
        if (err)
            return console.log(err);
        if (temp < desiredTemp) {
            exec('sudo ~/433Utils/RPi_utils/codesend 1135923')
        }
        if (temp > desiredTemp) {
            exec('sudo ~/433Utils/RPi_utils/codesend 1135932')
        }
    });
});
```

Kuva 17. Cron-lämpötilatehtävä.

Kuvassa 19 esitellään toteutusta, jota cronin luoma tehtävä kutsuu viiden minuutin välein. Tietokannasta haetaan käyttäjän syöttämä lämpötila ja sitä verrataan lämpömittarin lukemaan. Vertailun tuloksena lämpöpatteri laitetaan joko päälle tai pois päältä lähettämällä infrapunasignaali.

5 Android-sovellus

Android-sovelluksen idea oli tarjota helppo pääsy AngularJS -asiakassovelluksen käyttöliittymälle ilman, että käyttäjän tarvitsee erikseen avata selainta ja syöttää URL. Tähän

käytettiin androidin WebView -komponenttia, jonka avulla sovelluksen sisään voi implementoida selaimen. Komponentti liitetään sovelluksen main -näkymään ja sille sallitaan javascript, joka on oletuksena pois päältä.

Sovelluksen käynnistyessä WebView lähettää http POST -pyynnön Node.js -palvelimelle ja välittää käyttäjätunnukset pyynnön mukana. Näin käyttäjän ei tarvitse joka kerta syöttää käyttäjätunnuksia sovelluksen käynnistyessä. Lähetettävät käyttäjätunnukset säilytetään sovelluksen muistissa, ja jos niitä ei löydy, niin POST-pyyntö lähettää tyhjät tunnukset, ja palvelin ohjaa käyttäjän automaattisesti sisäänkirjautumissivulle. Sisäänkirjautumistiedot voi tallentaa laitteen muistiin sovelluksen sovellusvalikosta.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.webview);

    sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);
    userName = sharedPreferences.getString("username", "");
    password = sharedPreferences.getString("password", "");
    Log.d(TAG, userName +"   " +password);
    postData = ("username=" + userName + "&password=" + password);
    url = "http://192.168.10.113:50011/login";

    webview = (WebView) findViewById(R.id.webView);
    webview.setWebViewClient(new WebViewClient());
    webSettings = webview.getSettings();
    webSettings.setJavaScriptEnabled(true);
    webview.postUrl(url, postData.getBytes(Charset.forName("UTF-8")));
```

Kuva 18. WebView komponentin luominen sovelluksen onCreate metodissa.

Sovellukseen toteutettiin myös Google Cloud Messaging -palvelu, joka käynnistyy sovelluksen käynnistymisen yhteydessä ja jää pyörimään taustalle, vaikka sovellus suljettaisiin.

```
    if (checkPlayServices()) {
        // Start IntentService to register this application with GCM.
        Intent intent = new Intent(this, RegistrationService.class);
        startService(intent);
    }
}
```

Kuva 19. Gcm-palvelun käynnistys.

Sovelluksen onCreate-funktiossa luodaan Intent. Intentit ovat Androidin palveluja, joilla voidaan käynnistää uusi toiminnollisuus. Luotu Intent kutsuu RegistrationService-luokkaa service-tyyppisesti, eli kaikki sen toiminnot tapahtuvat sovelluksen taustalla.

RegistrationService-luokassa tehdään instanceID.getToken() -kutsu, jolle annetaan parametrina sender id. Kutsu palauttaa sovelluksen yksilöllisen tokenin. Token täytyy tämän jälkeen lähettää Node.js-sovelluspalvelimelle, jotta se pystyy kommunikoimaan Google Cloud Message -palvelun kanssa. Tokenin lähetys toteutettiin http POST-pyyntönä, jossa token välitettiin url-enkoodattuna (urlencoded) parametrina.

```
public RegistrationService() { super(TAG); }

@Override
protected void onHandleIntent(Intent intent) {
    SharedPreferences sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);

    try {

        InstanceID instanceID = InstanceID.getInstance(this);
        String token = instanceID.getToken(getString(R.string.gcm_defaultSenderId),
                GoogleCloudMessaging.INSTANCE_ID_SCOPE, null);
        // [END get_token]
        Log.i(TAG, "GCM Registration Token: " + token);
        HashMap<String, String> map = new HashMap<~>();
        map.put("token", token);
        //send token to server
        sendRegistrationToServer(map);

        sharedPreferences.edit().putBoolean(QuickStartPreferences.
                SENT_TOKEN_TO_SERVER, true).apply();
```

Kuva 20. Registration-luokan toteutus.

```
URL url = new URL("http://homecontrol.dy.fi:3000/token");
conn = (HttpURLConnection) url.openConnection();
conn.setConnectTimeout(15000);
conn.setRequestMethod("POST");
conn.setDoInput(true);
conn.setDoOutput(true);

conn.setRequestProperty("Content-Type", "application/json");
conn.setRequestProperty("Accept", "application/json");
conn.setRequestProperty("Content-Language", "en-US");

OutputStream os = conn.getOutputStream();
OutputStream out = new BufferedOutputStream(conn.getOutputStream());
BufferedWriter writer = new BufferedWriter(
        new OutputStreamWriter(os, "UTF-8"));
writer.write(getPostDataString(postDataParams));

writer.flush();
writer.close();
os.close();
```

Kuva 21. Tokenin lähetys palvelimelle.

Gcm-palvelun viestien vastaanottamista varten piti toteuttaa MyGcmListenerService-luokka, joka perii GcmListenerService-luokan, jotta sovellukselle tulevat viestit voidaan vastaanottaa. MyGcmListenerService ylikirjoittaa (override) perityn luokan OnMessage-Received-metodin, jota kutsutaan aina, kun sovellus vastaanottaa viestin. GcmListener-Service toimii sovelluksen taustalla, joten GCM-palvelimelta saapuvat viestit tulevat perille, vaikka itse sovellus ei ole päällä.

```
public class MyGcmListenerService extends GcmListenerService {

    private static final String TAG = "MyGcmListenerService";

    @Override
    public void onMessageReceived(String from, Bundle data) {
        String message = data.getString("data");
        Log.d(TAG, "From: " + from);
        Log.d(TAG, "Message: " + message);

        sendNotification(message);

    }
```

Kuva 22. MyGcmListenerService-luokan toteutus.

Ylikirjoitetussa metodissa kutsutaan sendNotification()-funktiota, jossa luodaan Adroid-notifikaatio. Notifikaatio näyttää sovelluspalvelimelta tulevan viestin Android-laitteen näytöllä ja ilmoitusnäytöllä (notification area).

6 Käyttöliittymänäkymä

Sovelluksen käyttöliittymän toteutuksessa on käytetty AngularJS:ää ja materializea. Tarkoituksena oli luoda helppokäyttöinen ja responsiivinen käyttöliittymä, jotta sitä olisi helppo käyttää ilman turhaa navigointia. Siihen sisällytettiin navigointipalkki, joka mobiililaitteella selattaessa tiivistyy ikonin alle

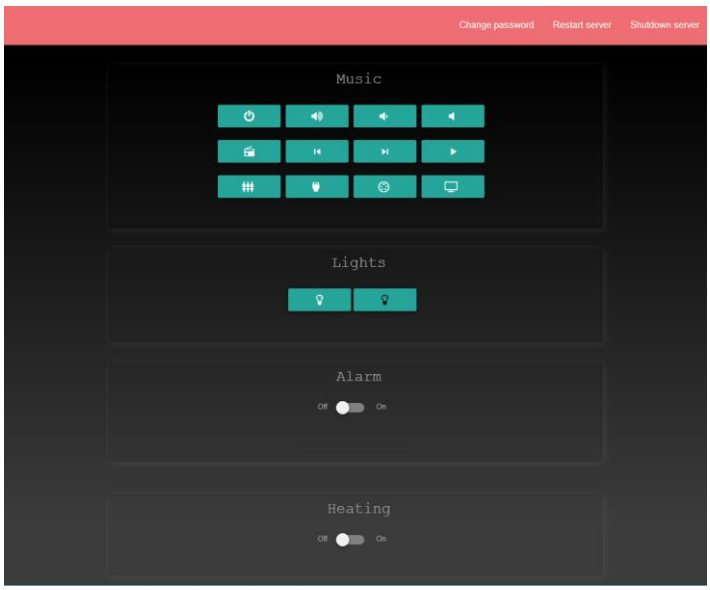
Materialize perustuu Googlen luomaan Material desingniin, jonka tavoitteen on luoda yhtenäinen käyttökokemus alustasta ja laitteesta rippumatta. Material desingin tarjoaa intuitiivisen käyttöliittymän, jossa elementtejä voi korostaa varjoilla, syvyysvaikutelmilla ja responsiivisilla animaatioilla.

Toteutetun käyttöliittymän painikkeet kuvainnollistavat haluttua toimintoa ikonein, ja tekstiä onkin käytetty vain otsakkeissa, joiden alle toiminnat on ryhmitelty.

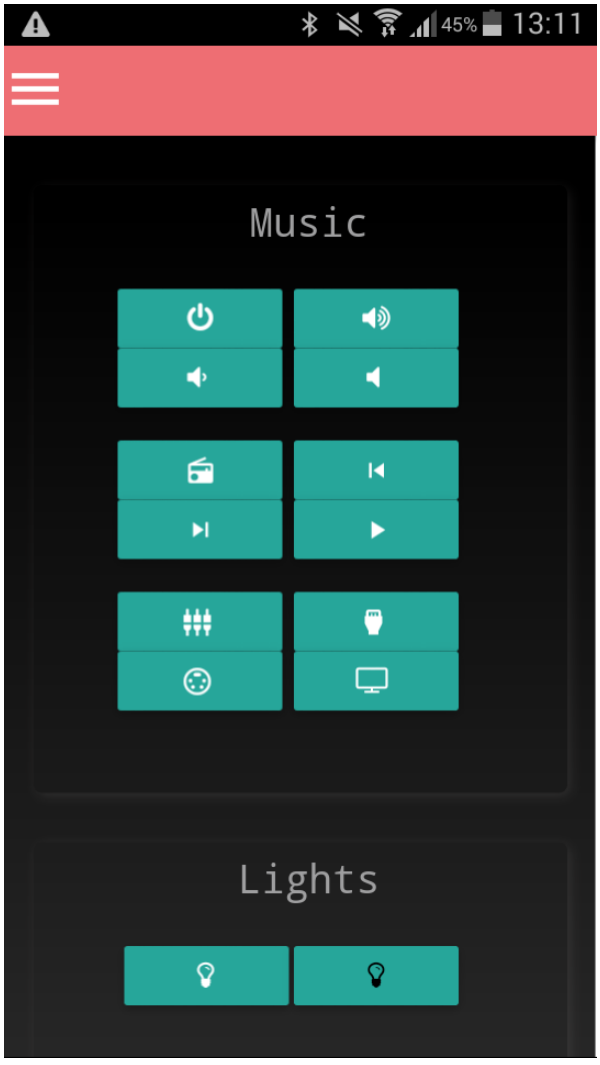
```
2  <div class="container about-content">
3      <h3>Music</h3>
4      <div class="row">
5          <div class="col s6 m6">
6              <a class="waves-effect waves-light btn" ng-click="power()"><i class="fa fa-
power-off"></i></a>
```

Kuva 23. Music-painikeryhmä ja power-painike.

Materializen waves-effect waves-light btn -elementillä on mahdollista luoda responsiivinen painike, joka reagoi, kun osoitin tuodaan sen ylle ja kun sitä painetaan. Painikkeen toiminta on sidottu angularin ng-click -direktiiviin.



Kuva 24. Käyttöliittymänäkymä selaimessa.



Kuva 25. Sovelluksen mobiilinäkymä.

7 Kehitysideat

Tutkimusyhtiö Gartnerin mukaan vuoden 2015 loppuun mennessä internettiin kytkettyjä laitteita on 4,9 miljardia kappaletta [13]. Hewlett Packard teetti tutkimuksen, jossa selvitettiin 25 yleisimmän IoT-laitteen tietoturvaa. Tutkimuksen lopputuleman mukaan 70 % tutkituista laitteista oli alttiita tietoturvauhkille. [14] Tietoturvauhat luovat aina taloudellisen vahingon vaaran sekä tahraavat brändin imagon [15].

Pi:lle toteutettua järjestelmää pystytään käyttämään kodin lähiverkossa sekä julkisessa internetissä. Internetiin kytkeytyminen luo kuitenkin aina turvallisuusuhan, varsinkin kun sillä voi ohjailla kodin laitteita. Pi:lle konfiguroitiin palomuuri sekä IP-osoitteiden lokittaminen ja lisääminen estolistalle, mikäli niistä tulee epäilyttävää verkkoliikennettä. Sovelluksen tietoturvaan ei otettu tässä työssä kantaa, vaan sitä on tarkoitus parantaa tulevaisuudessa, mm. salaamalla tietoliikenne ja validoimalla syötteet.

Pi:hin voi liittää myös WiFi-usb adapterin. WiFin avulla Pi pystyisi siirtämään suuriakin datamääriä verkossa olevien laitteiden kanssa, esim. valvontakameran kuvaa. Markkinoilla on saatavilla edullisia ja ohjelmoitavia WiFi-moduuleita, joiden avulla on mahdollista liittää esineitä verkkoon. Näiden avulla olisi mahdollista lisätä esim. kosteusmittarit kriittisiin tiloihin ja ohjelmoida ne lähettämään http-kutsu Pi:lle, mikäli vesivahinko pääsee syntymään.

8 Yhteenveto

Insinöörityön tavoitteina oli tutkia nykyaikaisia kodinohjausteknologioita ja kehittää kodinohjausprototyyppi, jota olisi helppo jatkokehittää tulevaisuudessa. Modulaarisuuden ansiosta sovellusta voidaan kehittää eteenpäin, esim. lisätä tuki erilaisille Gcm-viesteille ja ulkoisten laitteiden kytkennälle.

Raspberry Pi -minitietokoneelle rakennettiin toimiva ratkaisu, joka käyttää infrapunaa, radioaaltoja ja ohjausvirtaa ulkoisten laitteiden hallintaan. Pi:lle toteutettu ohjelmisto ra-

kennettiin käyttäen MongoDB-tietokantaa, ExpressJS-, AngularJS- ja NodeJS -sovelluskehyksiä (MEAN-pinototeutus). Ohjelmisto on kevyt, ja se toimii Pi:llä ilman suoristuskykyongelmia.

Rakennetulla ohjelmisto- ja laitteistokokonaisuudella saadaan sama lopputulema kuin kaupallisilla tuotteilla, mutta murto-osalla niiden hinnasta. IoT -laitteiden lisääntyessä näkisin, että vastaavilla toteutuksilla olisi markkinasijaa. Pi:n kaltaisen ohjelmoitavan keskusyksikön käyttäminen mahdollistaa edullisen ja mukautuvan lopputuotteen rakentamisen. Edullisten komponenttien ansiosta kokoonpanoa voidaan kehittää erilaisilla lisälaitteilla ilman kustannusten suurta nousua.

Lähteet

1. Faqs, Raspberry Pi. Saatavilla: https://www.raspberrypi.org/help/faqs/#softwareOS [Viitattu 14.11.2015].
2. Andrew K. Dennis. 2013. Raspberry Pi Home Automation with Arduino. UK: Packt Publishing.
3. 5mm Infrared LED, T-1 ¾, Everlight, 2010. Saatavilla: http://www.everlight.com/file/ProductFile/201407061514569412.pdf, [Viittattu 16.11.2015].
4. BC546 / BC547 / BC548 / BC549 / BC550 NPN Epitaxial Silicon Transistor, Fairchild, 2014. Saatavilla: https://www.fairchildsemi.com/datasheets/BC/BC547.pdf, [Viitattu 15.10.2015].
5. DS18B20 Programmable Resolution 1-Wire Digital Thermometer, maxim integrated, 2008. https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf, [Viitattu 16.10.2015].
6. Horan, Brendan; Hows, David; Membrey, Peter; 2013. Practical Raspberry Pi. Appress.
7. Mastering MEAN: Introducing the MEAN stack, IBM, 2014. Saatavilla: http://www.ibm.com/developerworks/library/wa-mean1/index.html, [Viitattu 16.11.2015].
8. Cloud Messaging, Google, 2015. Saatavilla: https://developers.google.com/cloud-messaging/gcm, [Viitattu 16.11.2015].
9. Google Cloud Messaging, Google, 2015. Saatavilla: https://developers.google.com/cloud-messaging/server, [Viitattu 1.10.2015].
10. Try Cloud Messaging for Android, Google 2015. Saatavilla: https://developers.google.com/cloud-messaging/android/start, [Viitattu 1.11.2015].
11. Mardan, Azat; Gonthier, Francois-Denis; Elst, Peter. 2014. Pro Express.js; Master Express.js: The Node.js Framework For Your Web Development. Appress.
12. Hows, David; Steneker, Stephen ; Beaulne, Alexandre ; Plugge, Eelco ; Membrey, Peter, 2014. MongoDB Basics, Apress, 2014.
13. Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015, Gartner 2014. Saatavilla: https://www.gartner.com/newsroom/id/2905717, [Viitattu 16.11.2015].
14. HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack, HP 2014. Saatavilla: http://www8.hp.com/us/en/hp-news/press-release.html?id=1744676#.VkmvSuK2zK6, [Viittattu 10.11.2015].

15. Sony Begins to Tally Its Financial Loss From Hack: $15 Million and Counting, recode, 2015. https://recode.net/2015/02/04/sony-begins-to-tally-its-financial-loss-from-hack-15-million-and-counting/.